This assignment involves developing a banking application that generates credit card statements in PDF format, simulating a real-world development task. Here's a structured approach to help you navigate through the process:

**1. Research Phase (Week 1)**

**A. Statement Sample Collection:**

* **Objective:** Gather your assigned bank's credit card statement format.
* **Action:**
  + Perform a Google image search for "[Bank Name] credit card statement sample".
  + Visit the bank's official website for statement examples or documentation.
  + If your assigned bank offers PDFs or online banking access, review those to extract the required format.
* **Deliverable:** A sample of the statement with clear sections for customer information, transactions, rewards, and summary details.

**B. Database Design:**

* **Objective:** Create an efficient database schema for customer information, transactions, rewards, etc.
* **Action:**
  + Define tables: Customers, Accounts, Transactions, Rewards, Statement, etc.
  + Determine relationships: A customer can have multiple accounts, each with many transactions, etc.
  + For each table, define attributes such as customer names, account numbers, transaction dates, amounts, fees, etc.

**C. Schema Example:**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

FullName VARCHAR(255),

Address VARCHAR(255),

Email VARCHAR(100),

PhoneNumber VARCHAR(20)

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID),

CardNumber VARCHAR(16),

AccountBalance DECIMAL(18, 2),

MinimumPayment DECIMAL(18, 2),

DueDate DATE

);

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

AccountID INT FOREIGN KEY REFERENCES Accounts(AccountID),

TransactionDate DATE,

Amount DECIMAL(18, 2),

Description VARCHAR(255),

TransactionType VARCHAR(50)

);

**2. Development Phase (Weeks 2-3)**

**A. Database Connection:**

* **Objective:** Connect to the SQL Server database to extract necessary data.
* **Action:** Choose a language (Python, C#, Java), set up the database connection, and implement efficient query operations.
* **Example in Python using pyodbc:**

import pyodbc

def connect\_to\_db():

conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'

'SERVER=server\_name;'

'DATABASE=database\_name;'

'UID=username;'

'PWD=password')

return conn

**B. PDF Generation:**

* **Objective:** Generate professional PDF credit card statements that match the assigned bank's format.
* **Action:** Use libraries like ReportLab (Python), iText (Java), or PDFsharp (C#) to create PDF files.
* **Ensure:**
  + Include necessary sections like customer details, transactions, balance summary, rewards, etc.
  + Format the statement similar to the sample, ensuring consistent fonts, colors, and spacing.
  + Implement multi-page support if required.

from reportlab.lib.pagesizes import letter

from reportlab.pdfgen import canvas

def generate\_pdf(statement\_data, file\_name):

c = canvas.Canvas(file\_name, pagesize=letter)

c.drawString(100, 750, f"Customer Name: {statement\_data['customer\_name']}")

c.drawString(100, 735, f"Account Number: {statement\_data['account\_number']}")

c.drawString(100, 720, f"Balance: {statement\_data['balance']}")

# Additional sections for transactions, summary, etc.

c.save()

**C. Currency and Date Formatting:**

* Format all financial values according to regional standards (e.g., Malaysia Ringgit, Singapore Dollar, etc.).
* Handle dates in the correct format (e.g., dd-mm-yyyy for Malaysia, yyyy-mm-dd for Singapore).

**D. Bilingual Support:**

* Implement bilingual support by adding a language toggle and ensuring both languages display correctly in the statement.
* Example: Using Python with gettext for translation.

**3. Testing Phase (Week 4)**

**A. Test Case Development:**

* **Objective:** Test the application for various scenarios.
* **Action:** Develop and execute at least 15 test cases covering:
  + Correctness of financial calculations (balances, minimum payments, etc.).
  + Transaction volumes (few, many, or none).
  + Bilingual functionality.
  + Handling edge cases (e.g., statement periods spanning across months).
  + Unusual transaction descriptions and amounts.

**B. Example Test Cases:**

1. Test with zero balance.
2. Test with multiple transactions (debits, credits, fees).
3. Test with a bilingual statement (English/Malay, for example).
4. Test with large transactions (e.g., high-value purchases).

**4. Presentation Phase (Final Week)**

**A. Demo Preparation:**

* **Objective:** Prepare a 10-minute demo showcasing your solution.
* **Action:**
  + Display the generated PDFs.
  + Explain how your application works, highlighting important features such as multi-language support, PDF generation, and financial calculations.
  + Demonstrate how the program handles different scenarios (e.g., different currencies, edge cases).

**B. Documentation:**

* Create comprehensive documentation including:
  + **Code documentation** explaining key functions and classes.
  + **User guide** explaining how to use and set up the application.
  + **Test case documentation** with results.
  + **Sample PDFs** demonstrating the generated statements.

**C. Presentation Structure:**

* Introduction to the problem.
* Solution overview (including database schema, functionality, and technology stack).
* Demo of the generated PDF statements.
* Address questions and challenges faced during development.

**5. Evaluation Criteria**

* **Functionality (40%)**: Ensure the PDF closely matches the bank’s format and the database operations work smoothly.
* **Technical Implementation (25%)**: Maintain code quality, efficient queries, and ensure performance.
* **Testing (20%)**: Cover various scenarios and document the test results.
* **Presentation and Documentation (15%)**: Present the solution clearly and provide robust documentation.

**Common Challenges & Tips:**

1. **Statement Formatting**: Pay close attention to replicating the exact layout. Adjust fonts, margins, and spacing carefully to match the sample.
2. **Handling Multi-page Statements**: Ensure proper pagination when generating longer statements.
3. **Accurate Financial Calculations**: Double-check the formulas for calculating balances, due amounts, and minimum payments.
4. **Efficient Querying**: Optimize SQL queries for large datasets to ensure quick retrieval of data.

Good luck with your assignment! If you need help with any particular phase or have more specific questions, feel free to ask!